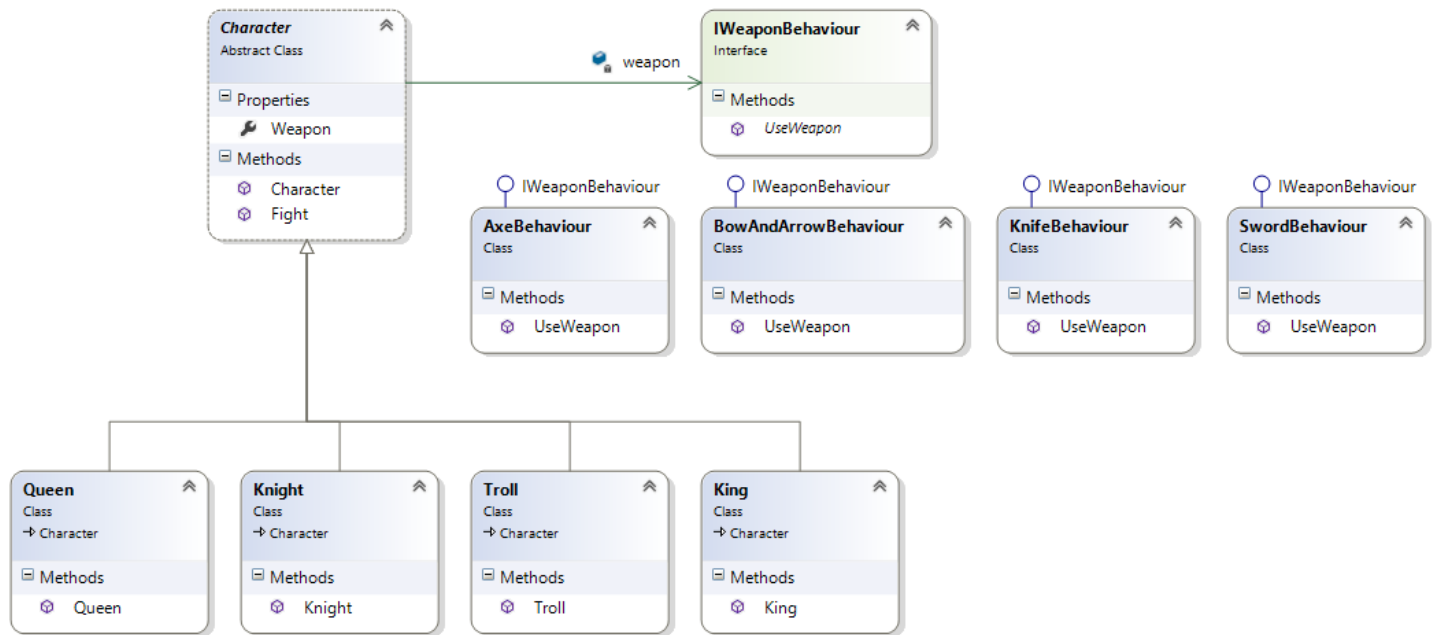


## Assignment 1 ('Strategy Pattern')

The following characters are present in a game: Queen, Knight, Troll and King. They all inherit from abstract base class 'Character'. Each character has a weapon to fight with. Create an interface `IWeaponBehaviour` and implement 4 different kind of weapons: Axe, BowAndArrow, Knife and Sword (all implementing interface `IWeaponBehaviour`). The class diagram below shows all classes/interfaces.



Each character has a default weapon, but this can change during the game (to another weapon).

Implement the classes/interfaces shown, and use the following main program to test it:

```

static void Main(string[] args)
{
    List<Character> characters = new List<Character>();
    characters.Add(new Queen());
    characters.Add(new Troll());
    characters.Add(new King());
    characters.Add(new Knight());

    foreach (Character character in characters)
        character.Fight();
    Console.WriteLine();

    // change weapon of knight to axe
    characters[3].Weapon = new AxeBehaviour();

    foreach (Character character in characters)
        character.Fight();

    Console.ReadKey();
}
  
```

This code has the following output →

```

file:///C:/Users/Gerwin...
Cutting with a knife
Chopping with an axe
Shooting an arrow with a bow
Swinging a sword

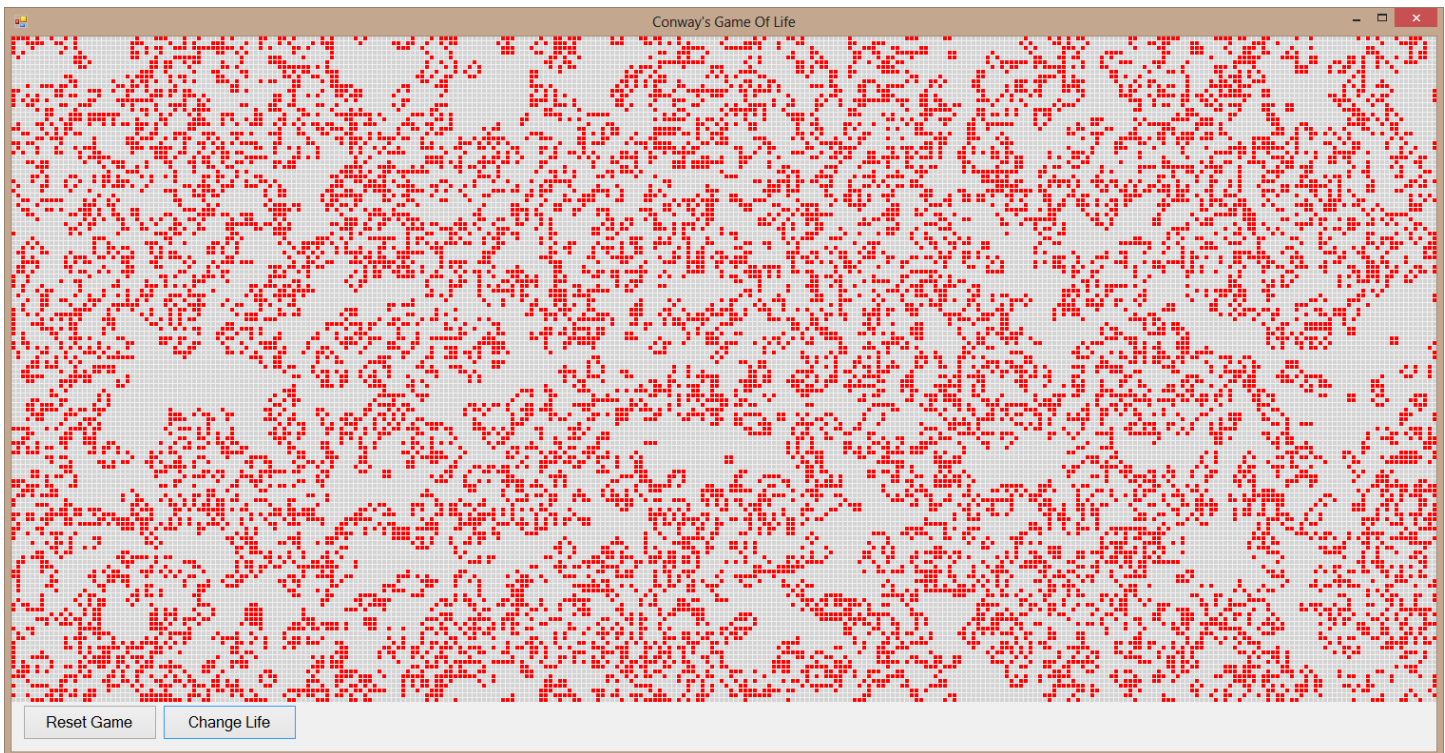
Cutting with a knife
Chopping with an axe
Shooting an arrow with a bow
Chopping with an axe
  
```

The screenshot shows a console window with the title 'file:///C:/Users/Gerwin...'. It displays the output of the program. The first four lines correspond to the initial weapons of the Queen, Knight, Troll, and King respectively. After a blank line, the next four lines show the output after the Knight's weapon has been changed to an axe, with the 'Shooting an arrow with a bow' line replaced by 'Chopping with an axe'.

## Assignment 2 ('Strategy Pattern')

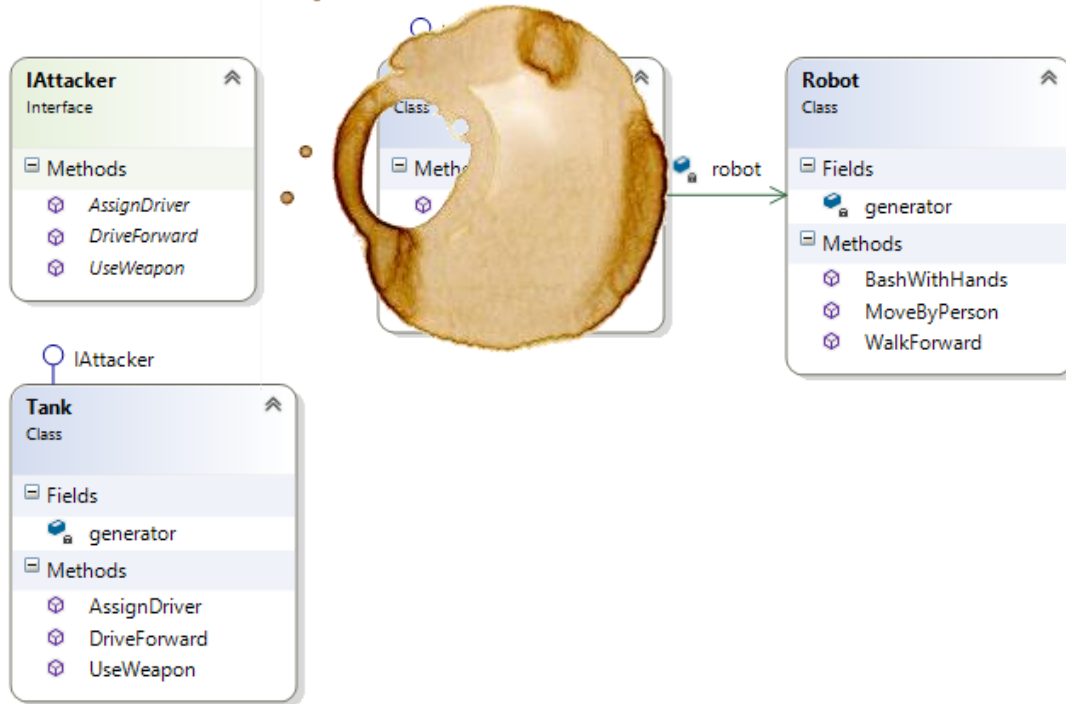
On Blackboard ('Week 4 assignments') you can find application 'Game of Life' (the same application as week 2). This application uses the default rules of life according to John Conway (B3/S23), see screenshot below. Change this application in order to have 2 different variants: a 'standard life' and a 'high life' variant (B36/S23, see <http://www.conwaylife.com/wiki/HighLife>). So, the same assignment as in week 2, but now implement the 2 variants by using the Strategy pattern on class `ConwayGameOfLife`, so the behaviour can be changed dynamically (without creating a new object). Method 'CellShouldLive' (called by method 'Evolve') must be implemented by 2 separate classes (StandardLife and HighLife), but now by implementing an interface (ILifeBehaviour).

For more information about Conway's Game of Life, see [https://en.wikipedia.org/wiki/Conway%27s\\_Game\\_of\\_Life](https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life).



## Assignment 3 ('Adapter Pattern')

In a very violent game (not suitable for students under 18 years old...) several 'attackers' are used, to defeat enemies. An example of an attacker is a Tank, implementing interface IAttacker. In the game also a Robot must be added, although it's not really an attacker. The solution for this is to create an adapter for this robot, as shown in the class diagram below (unfortunately someone spilled some coffee on this diagram...).



Create an application that implements the above classes/interfaces. Use the main program below, that produces the given output.

```

static void Main(string[] args)
{
    // create a tank (and assign it to a driver)
    // ...

    // create a robot (and let it move by a person)
    // ...

    // create attackers list, and add tank and robot
    List<IAttacker> attackers = new List<IAttacker>();
    // ...

    // process all attackers
    foreach (IAttacker attacker in attackers)
    {
        attacker.DriveForward();
        attacker.UseWeapon();
    }
}
  
```

```

file:///C:/Users/Gerwin ...
Frank is steering the tank
Robot is moved by Mark

Tank moves 3 positions forward
Tank causes damage

Robot walks 3 steps forward
Robot causes damage with hands
  
```